Should there be a community differentiated BLAS library?

Paul D. Hovland, Jean Utke
Mathematics & Computer Science Division
Argonne National Laboratory
Background: BLAS=basic linear algebra subprograms

- L1 = scalar, vector, vector-vector operations (15)
- L2 = matrix-vector operations (25)
- L3 = matrix-matrix operations (9)
- Variations on precision and real vs complex -> total of 150 subroutines and functions in F77
- F77 reference implementation (slow)
- Vendor specific implementations, ATLAS, Goto are optimized for performance
**Background: LAPACK = linear algebra package**

- linear system solvers, also for least-squares solutions, eigen/singular value problems
- reference implementation:
  - latest version on netlib: 3.2 (Nov/18/2008)
  - refers to BLAS
- vendor implementations optimized for speed
- with type/precision variations 1.5k routines (400+ of which are marked auxiliary)
Main Issue

- Many (?) codes make use of the BLAS (and LAPACK)
- Option #1: apply an AD tool to a reference implementation
  - Problem #1: semantics may not be correct, especially for complex
  - Problem #2: performance may be very poor
    - Differentiation interferes with optimizations in reference implementation
    - Derivatives may involve higher order tensors, providing opportunity for improved performance
- Option #2: develop a differentiated version of the BLAS by hand
  - Problem #1: a lot of work
  - Problem #2: noncomposability of the BLAS
  - Problem #3: need to support many interfaces
    - Floating point type
    - Activity status of parameters
    - Language
    - Derivative representation
other interfaces

- for BLAS
  - F95: INTEL's MKL, NAG's “F95 BLAS proposal”,...
  - C: in cblas
- for LAPACK:
  - F95: lapack95; C: clapack; Java: jlapack
- Issue: self contained implementation or just a wrapper calling something else (e.g. fast vendor libraries)?
- extensions: lapack++, xblas
a few issues

- many variations to be generated based on type (real, complex) and precision (single, double, ...) of the arguments
- even more variations based on the kind of activity pattern at the call site (e.g. in $Ax=b$, is $A$ active or only $b$ and $x$)
- comply with the F77/F95/??? interfaces?
- Is it worth the effort? (e.g. using blas vs. built in Fortran array ops)
wishful thinking

- have a place to contribute analytic implementation dX for derivatives of routine X
- tool knows how to interface its generated version ad_X with the analytic dX
- user grabs “ADified” blas/lapack from the common place and can link it for their tool
- really wishful: specify semantics of X and dX at high-level (e.g. matlab) and generate the code (e.g. with Matlab D)
AD of the reference impl.?

- might serve as a starting point **BUT**
- ultimate goal should be to treat BLAS/LAPACK routines as high-level intrinsics – why?
  - because the math of the routine is known we can provide an explicit (analytic) derivative
  - expect to gain performance advantage from explicit derivatives: example ...
  - example with `dgesvd` from Bastani/Guerrieri shows naïve differentiation may not satisfy expected properties (here certain matrix products being antisymmetric)
...also

- BLAS reference implementation is known to be slow (contains some “manual” code optimizations that can turn counterproductive)
- that sets up the derivatives to be slow too
- example ... (your dgemv/dgemm example)
- LAPACK to BLAS calls use a lot of difficult-to-analyze offsets into work arrays ....
- does not provide an implementation reusable for various tools